Kerberos Working Group G. Hudson, Ed.
Internet-Draft MIT Kerberos Consortium
Expires: April 8, 2012 October 6, 2011

Camellia Encryption for Kerberos 5
draft-ietf-krb-wg-camellia-cts-00

Abstract

This document specifies two encryption types and two corresponding checksum types for the Kerberos cryptosystem suite. The new types use the Camellia block cipher in CBC-mode with ciphertext stealing and the CMAC algorithm for integrity protection.

Status of this Memo

Copyright Notice

1. Introduction

The Camellia block cipher, described in [RFC3713], has a 128-bit block size and a 128-bit, 192-bit, or 256-bit key size, similar to AES. This document specifies Kerberos encryption and checksum types for Camellia using 128-bit or 256-bit keys. The new types conform to the framework specified in [RFC3961], but do not use the simplified profile.

Like the simplified profile, the new types use key derivation to produce keys for encryption, integrity protection, and checksum operations. Instead of the [RFC3961] section 5.1 key derivation function, the new types use a key derivation function from the family specified in [SP800-108].

The new types use the CMAC algorithm for integrity protection and checksum operations. As a consequence, they do not rely on a hash algorithm except when generating keys from strings.

Like the AES encryption types [RFC3962], the new encryption types use CBC mode with ciphertext stealing to avoid the need for padding. They also use the same PBKDF2 algorithm for key generation from strings, with a modification to the salt string to ensure that different keys are generated for Camellia and AES encryption types.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Protocol Key Representation

The Camellia key space is dense, so we use random octet strings directly as keys. The first bit of the Camellia bit string is the high bit of the first byte of the random octet string.

3. Key Generation from Strings

We use a variation on the key generation algorithm specified in [RFC3962] section 4.

First, to ensure that different long-term keys are used with Camellia and AES, we prepend the enctype name to the salt string, separated by a null byte. The enctype name is "camellia128-cts-cmac" or "camellia256-cts-cmac" (without the quotes).

Second, the final key derivation step uses the algorithm described in Section 4 instead of the key derivation algorithm used by the simplified profile.

Third, if no string-to-key parameters are specified, the default number of iterations is raised to 32768.

```
saltp = enctype-name | 0x00 | salt
tkey = random2key(PBKDF2-HMAC-SHA1(passphrase, saltp,
                                   iter_count, keylength))
key = KDF-FEEDBACK-CMAC(tkey, "kerberos")
```

4. Key Derivation

We use a key derivation function from the family specified in [SP800-108] section 5.2, "KDF in Feedback Mode". The PRF parameter of the key derivation function is CMAC with Camellia-128 or Camellia-256 as the underlying block cipher; this PRF has an output size of 128 bits. A block counter is used with a length of 4 bytes, represented in big-endian order. The length of the output key in bits (denoted as k) is also represented as a four-byte string in big-endian order. The label input to the KDF is the usage constant supplied to the key derivation function, and the context is unused.

```
n = ceiling(k / 128)
K0 = zeros
Ki = CMAC(key, K(i-1) | i | constant | 0x00 | k)
DR(key, constant) = k-truncate(K1 | K2 | ... | Kn)
KDF-FEEDBACK-CMAC(key, constant) = random-to-key(DR(key, constant))
```

The constants used for key derivation are the same as those used in the simplified profile.

5. CMAC Checksum Algorithm

For integrity protection and checksums, we use the CMAC function defined in [SP800-38B], with Camellia-128 or Camellia-256 as the underlying block cipher.

6. Kerberos Algorithm Protocol Parameters

The following parameters apply to the encryption types camellia128-cts-cmac, which uses a 128-bit protocol key, and camellia256-cts-cmac, which uses a 256-bit protocol key.

Protocol key format: as defined in Section 2.

Specific key structure: three protocol format keys: { Kc, Ke, Ki }.

Required checksum mechanism: as defined in Section 7.

Key generation seed length: the key size (128 or 256 bits).

String-to-key function: as defined in Section 3.

Default string-to-key parameters: 00 00 80 00.

Random-to-key function: identity function.

Key-derivation function: as indicated below, with usage represented as four octets in big-endian order.

```
Kc = KDF-FEEDBACK-CMAC(base-key, usage | 0x99)
Ke = KDF-FEEDBACK-CMAC(base-key, usage | 0xAA)
Ki = KDF-FEEDBACK-CMAC(base-key, usage | 0x55)
```

Cipher state: a 128-bit CBC initialization vector.

Initial cipher state: all bits zero.

Encryption function: as follows, where E() is Camellia encryption in CBC-CTS mode, with the next-to-last block used as the CBC-style ivec, or the last block if there is only one.

```
conf = Random string of 128 bits
(C, newstate) = E(Ke, conf | plaintext, oldstate)
M = CMAC(Ki, conf | plaintext)
ciphertext = C | M
```

Decryption function: as follows, where D() is Camellia decryption in CBC-CTS mode, with the ivec treated as in E().

```
(C, M) = ciphertext
(P, newIV) = D(Ke, C, oldstate)
if (M != CMAC(Ki, P)) report error
newstate = newIV
```

Pseudo-random function: as follows.

```
Kp = KDF-FEEDBACK-CMAC(protocol-key, "prf")
PRF = CMAC(Kp, octet-string)
```

7. Checksum Parameters

The following parameters apply to the checksum types cmac-camellia128 and cmac-camellia256, which are the associated checksum for camellia128-cts-cmac and camellia256-cts-cmac respectively.

Associated cryptosystem: Camellia-128 or Camellia-256 as appropriate for the checksum type.

get_mic: CMAC(Kc, message).

verify_mic: get_mic and compare.

8. Assigned Numbers

Encryption types

Type name	etype value	key size
camellia128-cts-cmac	TBD	128
camellia256-cts-cmac	TBD	256

Checksum types

Type name	sumtype value	length
cmac-camellia128	TBD	128
cmac-camellia256	TBD	128

9. Security Considerations

[CRYPTOENG] chapter 4 discusses weaknesses of the CBC cipher mode. An attacker who can observe enough messages generated with the same key to encounter a collision in ciphertext blocks could recover the XOR of the plaintexts of the previous blocks. Observing such a collision becomes likely as the number of blocks observed approaches

2^64. This consideration applies to all previously standardized Kerberos encryption types and all uses of CBC encryption with 128-bit block ciphers in other protocols. Kerberos deployments can mitigate this concern by rolling over keys often enough to make observing 2^64 messages unlikely.

Because the new checksum types are deterministic, an attacker could pre-compute checksums for a known plain-text message in 2^64 randomly chosen protocol keys. The attacker could then observe checksums legitimately computed in different keys until a collision with one of the pre-computed keys is observed; this becomes likely after the number of observed checksums approaches 2^64. Observing such a collision allows the attacker to recover the protocol key. This consideration applies to most previously standardized Kerberos checksum types and most uses of 128-bit checksums in other protocols.

Kerberos deployments should not migrate to the Camellia encryption types simply because they are newer, but should use them only if business needs require the use of Camellia, or if a serious flaw is discovered in AES which does not apply to Camellia.

The security considerations described in [RFC3962] section 8 regarding the string-to-key algorithm also apply to the Camellia encryption types.

At the time of writing this document, there are no known weak keys for Camellia, and no security problem has been found on Camellia (see [NESSIE], [CRYPTREC], and [LNCS5867]).

10. IANA Considerations

Assign two Kerberos Encryption Type Numbers for camellia128-cts-cmac and camellia256-cts-cmac.

Assign two Kerberos Checksum Type Numbers for cmac-camellia128 and camellia256.

11. Test Vectors

Sample results for string-to-key conversion:

```
Iteration count = 1
Pass phrase = "password"
Salt = "ATHENA.MIT.EDUraeburn"
128-bit Camellia key:
    57 D0 29 72 98 FF D9 D3 5D E5 A4 7F B4 BD E2 4B
```

```
   256-bit Camellia key:
       B9 D6 82 8B 20 56 B7 BE 65 6D 88 A1 23 B1 FA C6
       82 14 AC 2B 72 7E CF 5F 69 AF E0 C4 DF 2A 6D 2C

   Iteration count = 2
   Pass phrase = "password"
   Salt = "ATHENA.MIT.EDUraeburn"
   128-bit Camellia key:
       73 F1 B5 3A A0 F3 10 F9 3B 1D E8 CC AA 0C B1 52
   256-bit Camellia key:
       83 FC 58 66 E5 F8 F4 C6 F3 86 63 C6 5C 87 54 9F
       34 2B C4 7E D3 94 DC 9D 3C D4 D1 63 AD E3 75 E3

   Iteration count = 1200
   Pass phrase = "password"
   Salt = "ATHENA.MIT.EDUraeburn"
   128-bit Camellia key:
       8E 57 11 45 45 28 55 57 5F D9 16 E7 B0 44 87 AA
   256-bit Camellia key:
       77 F4 21 A6 F2 5E 13 83 95 E8 37 E5 D8 5D 38 5B
       4C 1B FD 77 2E 11 2C D9 20 8C E7 2A 53 0B 15 E6

   Iteration count = 5
   Pass phrase = "password"
   Salt=0x1234567878563412
   128-bit Camellia key:
       00 49 8F D9 16 BF C1 C2 B1 03 1C 17 08 01 B3 81
   256-bit Camellia key:
       11 08 3A 00 BD FE 6A 41 B2 F1 97 16 D6 20 2F 0A
       FA 94 28 9A FE 8B 27 A0 49 BD 28 B1 D7 6C 38 9A

   Iteration count = 1200
   Pass phrase = (64 characters)
     "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
   Salt="pass phrase equals block size"
   128-bit Camellia key:
       8B F6 C3 EF 70 9B 98 1D BB 58 5D 08 68 43 BE 05
   256-bit Camellia key:
       11 9F E2 A1 CB 0B 1B E0 10 B9 06 7A 73 DB 63 ED
       46 65 B4 E5 3A 98 D1 78 03 5D CF E8 43 A6 B9 B0

   Iteration count = 1200
   Pass phrase = (65 characters)
     "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
   Salt = "pass phrase exceeds block size"
   128-bit Camellia key:
       57 52 AC 8D 6A D1 CC FE 84 30 B3 12 87 1C 2F 74
   256-bit Camellia key:
```

```
        61 4D 5D FC 0B A6 D3 90 B4 12 B8 9A E4 D5 B0 88
        B6 12 B3 16 51 09 94 67 9D DB 43 83 C7 12 6D DF

    Iteration count = 50
    Pass phrase = g-clef (0xf09d849e)
    Salt = "EXAMPLE.COMpianist"
    128-bit Camellia key:
        CC 75 C7 FD 26 0F 1C 16 58 01 1F CC 0D 56 06 16
    256-bit Camellia key:
        16 3B 76 8C 6D B1 48 B4 EE C7 16 3D F5 AE D7 0E
        20 6B 68 CE C0 78 BC 06 9E D6 8A 7E D3 6B 1E CC

    Sample results for key derivation:

    128-bit Camellia key:
        57 D0 29 72 98 FF D9 D3 5D E5 A4 7F B4 BD E2 4B
    Kc value for key usage 2 (constant = 0x0000000299):
        D1 55 77 5A 20 9D 05 F0 2B 38 D4 2A 38 9E 5A 56
    Ke value for key usage 2 (constant = 0x00000002AA):
        64 DF 83 F8 5A 53 2F 17 57 7D 8C 37 03 57 96 AB
    Ki value for key usage 2 (constant = 0x0000000255):
        3E 4F BD F3 0F B8 25 9C 42 5C B6 C9 6F 1F 46 35

    256-bit Camellia key:
        B9 D6 82 8B 20 56 B7 BE 65 6D 88 A1 23 B1 FA C6
        82 14 AC 2B 72 7E CF 5F 69 AF E0 C4 DF 2A 6D 2C
    Kc value for key usage 2 (constant = 0x0000000299):
        E4 67 F9 A9 55 2B C7 D3 15 5A 62 20 AF 9C 19 22
        0E EE D4 FF 78 B0 D1 E6 A1 54 49 91 46 1A 9E 50
    Ke value for key usage 2 (constant = 0x00000002AA):
        41 2A EF C3 62 A7 28 5F C3 96 6C 6A 51 81 E7 60
        5A E6 75 23 5B 6D 54 9F BF C9 AB 66 30 A4 C6 04
    Ki value for key usage 2 (constant = 0x0000000255):
        FA 62 4F A0 E5 23 99 3F A3 88 AE FD C6 7E 67 EB
        CD 8C 08 E8 A0 24 6B 1D 73 B0 D1 DD 9F C5 82 B0

    Sample encryptions (all using the default cipher state):

    Plaintext: (empty)
    128-bit Camellia key:
        1D C4 6A 8D 76 3F 4F 93 74 2B CB A3 38 75 76 C3
    Random confounder:
        B6 98 22 A1 9A 6B 09 C0 EB C8 55 7D 1F 1B 6C 0A
    Ciphertext:
        C4 66 F1 87 10 69 92 1E DB 7C 6F DE 24 4A 52 DB
        0B A1 0E DC 19 7B DB 80 06 65 8C A3 CC CE 6E B8

    Plaintext: 1
```

```
   Random confounder:
       6F 2F C3 C2 A1 66 FD 88 98 96 7A 83 DE 95 96 D9
   128-bit Camellia key:
       50 27 BC 23 1D 0F 3A 9D 23 33 3F 1C A6 FD BE 7C
   Ciphertext:
       84 2D 21 FD 95 03 11 C0 DD 46 4A 3F 4B E8 D6 DA
       88 A5 6D 55 9C 9B 47 D3 F9 A8 50 67 AF 66 15 59
       B8

   Plaintext: 9 bytesss
   Random confounder:
       A5 B4 A7 1E 07 7A EE F9 3C 87 63 C1 8F DB 1F 10
   128-bit Camellia key:
       A1 BB 61 E8 05 F9 BA 6D DE 8F DB DD C0 5C DE A0
   Ciphertext:
       61 9F F0 72 E3 62 86 FF 0A 28 DE B3 A3 52 EC 0D
       0E DF 5C 51 60 D6 63 C9 01 75 8C CF 9D 1E D3 3D
       71 DB 8F 23 AA BF 83 48 A0

   Plaintext: 13 bytes byte
   Random confounder:
       19 FE E4 0D 81 0C 52 4B 5B 22 F0 18 74 C6 93 DA
   128-bit Camellia key:
       2C A2 7A 5F AF 55 32 24 45 06 43 4E 1C EF 66 76
   Ciphertext:
       B8 EC A3 16 7A E6 31 55 12 E5 9F 98 A7 C5 00 20
       5E 5F 63 FF 3B B3 89 AF 1C 41 A2 1D 64 0D 86 15
       C9 ED 3F BE B0 5A B6 AC B6 76 89 B5 EA

   Plaintext: 30 bytes bytes bytes bytes byt
   Random confounder:
       CA 7A 7A B4 BE 19 2D AB D6 03 50 6D B1 9C 39 E2
   128-bit Camellia key:
       78 24 F8 C1 6F 83 FF 35 4C 6B F7 51 5B 97 3F 43
   Ciphertext:
       A2 6A 39 05 A4 FF D5 81 6B 7B 1E 27 38 0D 08 09
       0C 8E C1 F3 04 49 6E 1A BD CD 2B DC D1 DF FC 66
       09 89 E1 17 A7 13 DD BB 57 A4 14 6C 15 87 CB A4
       35 66 65 59 1D 22 40 28 2F 58 42 B1 05 A5

   Plaintext: (empty)
   Random confounder:
       3C BB D2 B4 59 17 94 10 67 F9 65 99 BB 98 92 6C
   256-bit Camellia key:
       B6 1C 86 CC 4E 5D 27 57 54 5A D4 23 39 9F B7 03
       1E CA B9 13 CB B9 00 BD 7A 3C 6D D8 BF 92 01 5B
   Ciphertext:
       03 88 6D 03 31 0B 47 A6 D8 F0 6D 7B 94 D1 DD 83
```

```
        7E CC E3 15 EF 65 2A FF 62 08 59 D9 4A 25 92 66

    Plaintext: 1
    Random confounder:
        DE F4 87 FC EB E6 DE 63 46 D4 DA 45 21 BB A2 D2
    256-bit Camellia key:
        1B 97 FE 0A 19 0E 20 21 EB 30 75 3E 1B 6E 1E 77
        B0 75 4B 1D 68 46 10 35 58 64 10 49 63 46 38 33
    Ciphertext:
        2C 9C 15 70 13 3C 99 BF 6A 34 BC 1B 02 12 00 2F
        D1 94 33 87 49 DB 41 35 49 7A 34 7C FC D9 D1 8A
        12

    Plaintext: 9 bytesss
    Random confounder:
        AD 4F F9 04 D3 4E 55 53 84 B1 41 00 FC 46 5F 88
    256-bit Camellia key:
        32 16 4C 5B 43 4D 1D 15 38 E4 CF D9 BE 80 40 FE
        8C 4A C7 AC C4 B9 3D 33 14 D2 13 36 68 14 7A 05
    Ciphertext:
        9C 6D E7 5F 81 2D E7 ED 0D 28 B2 96 35 57 A1 15
        64 09 98 27 5B 0A F5 15 27 09 91 3F F5 2A 2A 9C
        8E 63 B8 72 F9 2E 64 C8 39

    Plaintext: 13 bytes byte
    Random confounder:
        CF 9B CA 6D F1 14 4E 0C 0A F9 B8 F3 4C 90 D5 14
    256-bit Camellia key:
        B0 38 B1 32 CD 8E 06 61 22 67 FA B7 17 00 66 D8
        8A EC CB A0 B7 44 BF C6 0D C8 9B CA 18 2D 07 15
    Ciphertext:
        EE EC 85 A9 81 3C DC 53 67 72 AB 9B 42 DE FC 57
        06 F7 26 E9 75 DD E0 5A 87 EB 54 06 EA 32 4C A1
        85 C9 98 6B 42 AA BE 79 4B 84 82 1B EE

    Plaintext: 30 bytes bytes bytes bytes byt
    Random confounder:
        64 4D EF 38 DA 35 00 72 75 87 8D 21 68 55 E2 28
    256-bit Camellia key:
        CC FC D3 49 BF 4C 66 77 E8 6E 4B 02 B8 EA B9 24
        A5 46 AC 73 1C F9 BF 69 89 B9 96 E7 D6 BF BB A7
    Ciphertext:
        0E 44 68 09 85 85 5F 2D 1F 18 12 52 9C A8 3B FD
        8E 34 9D E6 FD 9A DA 0B AA A0 48 D6 8E 26 5F EB
        F3 4A D1 25 5A 34 49 99 AD 37 14 68 87 A6 C6 84
        57 31 AC 7F 46 37 6A 05 04 CD 06 57 14 74

    Sample checksums:
```

```
Plaintext: abcdefghijk
Checksum type: cmac-camellia128
128-bit Camellia key:
    1D C4 6A 8D 76 3F 4F 93 74 2B CB A3 38 75 76 C3
Key usage: 7
Checksum:
    11 78 E6 C5 C4 7A 8C 1A E0 C4 B9 C7 D4 EB 7B 6B

Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Checksum type: cmac-camellia128
128-bit Camellia key:
    50 27 BC 23 1D 0F 3A 9D 23 33 3F 1C A6 FD BE 7C
Key usage: 8
Checksum:
    D1 B3 4F 70 04 A7 31 F2 3A 0C 00 BF 6C 3F 75 3A

Plaintext: 123456789
Checksum type: cmac-camellia256
256-bit Camellia key:
    B6 1C 86 CC 4E 5D 27 57 54 5A D4 23 39 9F B7 03
    1E CA B9 13 CB B9 00 BD 7A 3C 6D D8 BF 92 01 5B
Key usage: 9
Checksum:
    87 A1 2C FD 2B 96 21 48 10 F0 1C 82 6E 77 44 B1

Plaintext: !@#$%^&*()!@#$%^&*()!@#$%^&*()
Checksum type: cmac-camellia256
256-bit Camellia key:
    32 16 4C 5B 43 4D 1D 15 38 E4 CF D9 BE 80 40 FE
    8C 4A C7 AC C4 B9 3D 33 14 D2 13 36 68 14 7A 05
Key usage: 10
Checksum:
    3F A0 B4 23 55 E5 2B 18 91 87 29 4A A2 52 AB 64
```

12. References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3713] Matsui, M., Nakajima, J., and S. Moriai, "A Description of the Camellia Encryption Algorithm", RFC 3713, April 2004.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

[RFC3962] Raeburn, K., "Advanced Encryption Standard (AES) Encryption for Kerberos 5", RFC 3962, February 2005.

[SP800-38B]
Dwork in, M., "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication", NIST Special Publication 800-38B, October 2009.

[SP800-108]
Chen, L., "Recommendation for Key Derivation Using Pseudorandom Functions", NIST Special Publication 800-108, October 2009.

[CRYPTOENG]
Schneier, B., "Cryptography Engineering", March 2010.

[CRYPTREC]
Information-technology Promotion Agency (IPA), "Cryptography Research and Evaluation Committees", <http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html>.

[LNCS5867]
Mala, H., Shakiba, M., and M. Dakhil-alian, "New Results on Impossible Different Cryptanalysis of Reduced Round Camellia-128", LNCS 5867, November 2009, <http://www.springerlink.com/content/e55783u422436g77/>.

[NESSIE] The NESSIE Project, "New European Schemes for Signatures, Integrity, and Encryption", <http://www.cosic.esat.kuleuven.be/nessie/>.

Appendix A. Notes to RFC Editor

Change the "TBD" entries in Section 8 to the values assigned by IANA.

Remove this section.

Author's Address

Greg Hudson (editor)
MIT Kerberos Consortium

Email: ghudson@mit.edu